CHUYÊN ĐỀ
TỔ CHỨC DỮ LIỆU
CÁC KIỂU DỮ LIỆU NÂNG CAO 2
CẤU TRÚC

Giảng viên: VŨ QUỐC HOÀNG
(vqhoang@fit.hcmus.edu.vn)

Nội dung trình bày

- Kiểu dữ liệu phức hợp
- Kiểu cấu trúc
- Con trỏ cấu trúc
- Cấu trúc tự trỏ
- Kiểu hợp
- Một số vấn đề với cấu trúc

Kiểu dữ liệu phức hợp

- Kiểu dữ liệu phức hợp (composite datatype) là kiểu dữ liệu được xây dựng từ các kiểu dữ liệu khác, các kiểu đó có thể là:
  - kiểu dữ liệu cơ bản
  - hoặc, kiểu dữ liệu phức hợp khác
- Hai loại kiểu dữ liệu phức hợp quan trọng là:
  - Kiểu cấu trúc (struct)
  - Kiểu lớp (class)
- Những kiểu do người dùng định nghĩa thường là những kiểu dữ liệu phức hợp:
  - Ngôn ngữ cung cấp phương tiện để người dùng định nghĩa (khai báo) những kiểu này

Kiểu cấu trúc (struct)

- Kiểu cấu trúc là kiểu dữ liệu gồm nhiều thành phần dữ liệu:
  - Các thành phần có thể khác kiểu nhau
  - Các thành phần được lưu trữ liên tiếp nhau trên vùng nhớ
    - Vùng nhớ lưu trữ một cấu trúc là tổng vùng nhớ lưu trữ các thành phần (và các vùng đệm nếu cần)
  - Các thành phần có thể có kiểu là kiểu cấu trúc khác
  - Kiểu cấu trúc còn được gọi là kiểu bản ghi (record); các thành phần còn được gọi là trường (field)

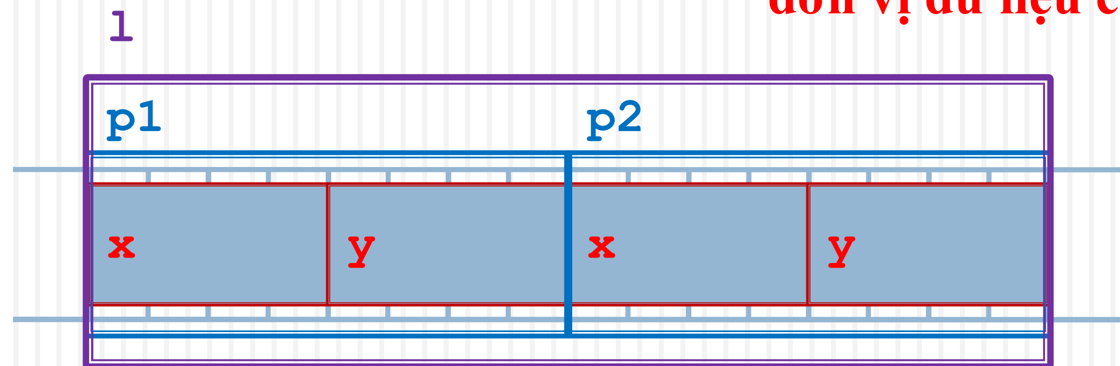
Kiểu cấu trúc


```
struct Point
{
        int x;
        int y;
};
```

```
struct Line
{
        Point p1;
        Point p2;
};
```

```
Line l;
```

struct cho phép nhóm các dữ liệu liên quan thành một đơn vị dữ liệu có nghĩa



Kiểu cấu trúc
Các thao tác cơ bản

- Khai báo kiểu

```
struct Point
{
    int x;
    int y;
};
```

Tên kiểu

Kiểu và tên các thành phần

- Khai báo biến

```
struct Point p; //C
Point q; //C++
```

- Khởi tạo

```
Point p = {10, 20};
```

	x:	y:
p:	10	20

Kiểu cấu trúc
Các thao tác cơ bản

- Phép gán

```
Point p = {10, 20};
Point q = p; //q={10, 20}
```

- Truy cập thành phần

```
Line l;
l.p1 = p;
l.p1.x = l.p2.x;
```

Con trỏ cấu trúc


```
Line l = {{10, 20}, {30, 40}};
Point p = {50, 60};
Point *pp = &p;
Point &rp = l.p1;
Line *pl = &l;
printf("%d, %d", pp->x, (*pp).y);//50, 60
printf("%d, %d", rp.x, rp.y); //10, 20
pp = &(pl->p1);
pp++;
printf("%d, %d", pp->x, (*pp).x);//30, 40
```

Ép kiểu cấu trúc

- C/C++ không cho phép chuyển kiểu trên kiểu cấu trúc

```
struct Int1
{
        int a;
};
struct Int2
{
        int a;
};

Int1 i1 = {10};
int i = i1.a; //ok
int j = (int)i1; //error
i1 = 10; //error
Int2 i2 = (Int2)i1; //error
```

Ép kiểu cấu trúc

- C/C++ cho phép chuyển kiểu tường minh trên kiểu con trỏ cấu trúc

```
struct Point        struct Line
{                   {
   int x, y;           Point p1, p2;
};                  };
```

```
Point p = {10, 20};
int *pi = (int*)&p;
(*pi) = 30;
printf("%d, %d", p.x, p.y); //30, 20
Line l = {{1, 2}, {3, 4}};
Line *pl = &l;
Point *pp = (Point*)l;
pp++;
printf("%d, %d", pp->x, pp->y); //3, 4
int ai[] = {5, 6, 7, 8};
pl = (Line*)ai;
printf("%d, %d", pl->p1.x, pl->p2.y); //5, 8
```

Cấu trúc tự trỏ

- Nhiều cấu trúc dữ liệu là cấu trúc đệ qui: cấu trúc có một hay nhiều thành phần có dạng của chính nó
- Cây nhị phân là một cấu trúc đệ qui gồm một nút mang dữ liệu và hai cây con trái phải (cũng là cây nhị phân)

```
struct BinTree
{
    int data;
    BinTree left, right; //!!!
};
```

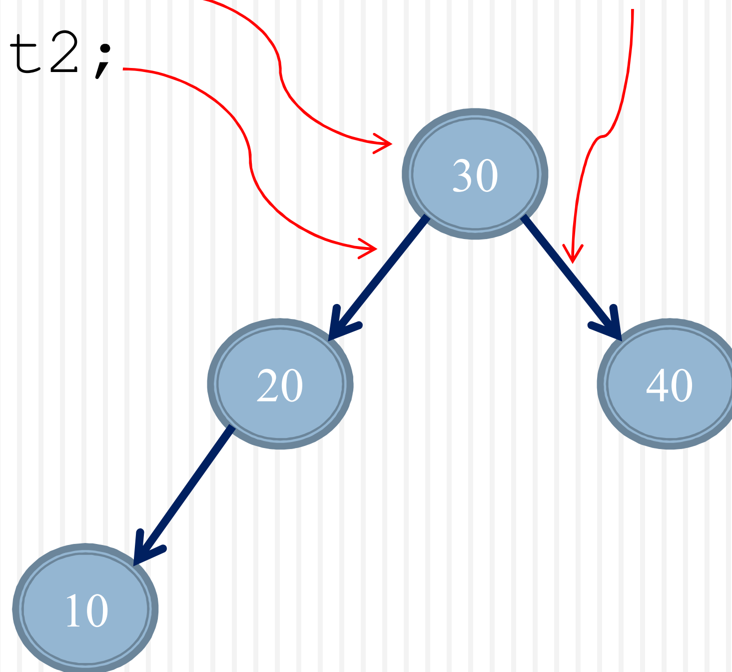
Cấu trúc tự trỏ

- Những cấu trúc đệ qui được hiện thực trong C/C++ bằng những cấu trúc tự trỏ

```
struct BinTree
{
    int data;
    BinTree *left, *right; //ok
};
```

Cấu trúc tự trỏ


```
BinTree t1 = {10, NULL, NULL};
BinTree t2 = {20, &t1, NULL};
BinTree t4 = {40, NULL, NULL};
BinTree t3 = {30, NULL, &t4};
t3.left = &t2;
```



Kiểu hợp (union)

- Kiểu hợp là kiểu dữ liệu gồm nhiều thành phần dữ liệu được lưu trữ tại cùng một vùng nhớ
- Kiểu hợp cho phép nhiều cách diễn giải (cách hiểu) trên cùng một dữ liệu
- Cú pháp như kiểu cấu trúc

Kiểu hợp


```
union XXX
{
   int is[4];
   Point ps[2];
   Line l;
};

XXX xxx = {10, 20, 30, 40};
printf("%d, %d", xxx.ps[0].x, xxx.ps[0].y); //10, 20
printf("%d, %d", xxx.l.p2.x, xxx.l.p2.y); //30, 40
xxx.l.p2 = xxx.l.p1;
printf("%d", xxx.is[3]); //20
```

Một số vấn đề với cấu trúc
Sao chép cấu trúc

- Thao tác sao chép cấu trúc là sao chép toàn bộ và chỉ vùng nhớ của cấu trúc

```
struct X
{
    char *s;
    int i;
};

char a[] = "hi";
X x1 = {a, 1};
X x2 = x1;
printf("%s, %d", x1.s, x1.i); //hi, 1
x2.i = 2; x2.s[0] = 'H';
printf("%s, %d", x1.s, x1.i); //Hi, 1
```

Một số vấn đề với cấu trúc Kiểu mở rộng

- Kiểu cấu trúc *A* được gọi là kiểu mở rộng của kiểu cấu trúc *B* nếu *A* có đầy đủ và đúng thứ tự các thành phần của *B* (và có thể có thêm các thành phần khác)
  - Là một dạng đặc biệt của kiểu con (subtype)
  - Là một dạng kế thừa chức năng: các thao tác trên một kiểu cũng có thể dùng để thao tác trên các kiểu mở rộng của nó

Một số vấn đề với cấu trúc Kiểu mở rộng


```
struct X
{
      int a, b;
};
struct XX
{
      int c, d;
      int e;
};
void f(X *x)
{
      x->a = x->b;
}

XX xx = {1, 2, 3};
f((X*)&xx);
printf("%d, %d, %d", x.c, xx.d, xx.e); //2, 2, 3
```

Hỏi và Đáp